أساسيات اللغة

أسئلة وتمارين

أسئلة

1. عندما تقوم بتجميع برنامج مكتوب بلغة البرمجة جافا، فإن المُجمع يحول ملف الشيفرة المفهومة من البشر إلى شيفرة مفهومة من طرف الآلة الإفتراضية ومستقلة عن المنصة. ماذا نسمي هذه الشيفرة؟
2. أي واحد من الإقتراحات التالية ليس تعليقا صحيحا :
   a. /** comment */
   b. /* comment */
   c. /* comment
   d. // comment
3. ما هو أول شيء يجب أن تتحقق منه عند حصولك على الخطأ التالي أثناء التشغيل :

```
Exception in thread "main" java.lang.NoClassDefFoundError: HelloWorldApp.
```

4. ما هو التوقيع الصحيح للطريقةmain ؟
5. عند تعريف الطريقةmain ، أي مُحول يجب أن يكتب أولا، publicأوstatic ؟
6. ما هو المُعطى التي تُعَرفه الطريقةmain ؟

تمارين

1. عَدِّل البرنامج HelloWorldApp.java بحيث تتم طباعة Hola Mundo! بدل.Hello World!
2. ستجد نسخة مُعَدلة قليلا من HelloWorldApp هنا: HelloWorldApp2.java . البرنامج يحتوي على خطأ. أصلح الخطأ بحيث تنجح عملية تجميع وتشغيل البرنامج. ماذا كان الخطأ؟

الأجوبة

أجوبة الأسئلة

1. البايت كود.Bytecode
2. الخيار الثالث هو التعليق غير الصحيح.
3. تحقق من الـclasspath، لأن المشغل لم يجد فِئَتك.
4. التوقيع الصحيح هو

```
public static void main(String[] args)
```

5. كلا المحولين يمكن أن يكون أولا أو ثانيا، لكن الإجماع أن يكون public أولا.public static :
6. الطريقة main تعرف مُعطى واحدا، يُسَمى عادة"args" ، و هو مصفوفة من كائنات.String

حلول التمارين

1. هذا هو السطر الوحيد الذي يجب تغييره :

```
System.out.println("Hola Mundo!"); //Display the string.
```

2. هذه هي الأخطاء التي ستحصل عليها عند محاولة تجميع البرنامج :

```
HelloWorldApp2.java:40: unclosed string literal
        System.out.println("Hello World!); // Display the string.
```

```
                              ^
HelloWorldApp2.java:40: ';' expected
        System.out.println("Hello World!); // Display the string.
                                                                  ^
HelloWorldApp2.java:42: reached end of file while parsing
}
 ^
3 errors
```

لتصحيح الخطأ، يجب إغلاق علامة الإقتباس المحيطة بالنص. السطر التالي يقوم بتصحيح الخطأ :

```
System.out.println("Hello World!"); //Display the string.
```

المتغيرات

تعلمت في الدروس السابقة أن الكائن يخزن حالته في حقول.

```
int cadence = 0;
int speed = 0;
int gear = 1;
```

تعَرَّفْت على ماهية الحقول في درس ما هو الكائن؟، لكنك في الغالب مازالت لديك بعض التساؤلات: ما هي قواعد ومعايير تسمية الحقول؟ بالإضافة إلىint ، ماهي أنواع البيانات الأخرى؟ هل يجب تحديد قيمة الحقول عند إعلانها؟ هل يتم إسناد قيمة افتراضية للحقل إذا لم يتم تحديد قيمته؟ سنَتَحرَّى الأجوبة على هذه الأسئلة في هذا الدرس، لكن قبل ذلك، هناك بعض الأمور التقنية التي يجب تمييزها.

في لغة البرمجة جافا، يتم استعمال كلا المصطلحين "حقل" و "متغير"؛ هذا الأمر يسبب ارتباكا للمبرمجين الجدد، لأن المصطلحين، وفي كثير من الأحيان، يشيران لنفس الشيء .

لغة البرمجة جافا تُحدد أنواع المتغيرات التالية:

- متغيرات الكائن Instance Variables (Non-Static Fields) تخزن الكائنات حالتها في حقولnon-static ، أي حقول مُعَرَّفة بدون الكلمة المفتاحية static. الحقول non-static تُعْرف أيضا ب instance variables لأن قيمتها تكون فريدة في كل instance من الفئة (بتعبير آخر، في كل كائن)؛ قيمة currentSpeed الخاصة بدراجة ما مُسْتَقّلة تماما عن قيمة ال currentSpeedالخاصة بدراجة أخرى.
- متغيرات الفئة (Static Fields) متغير الفئة عبارة عن حقل مُعْلن باستعمال المُحَوِّلstatic ؛ هذا يعني للمجمع أنه توجد نسخة واحدة فقط من هذا المتغير، بِغَضِّ النظر عن عدد الكائنات المُنْشَأة .مثلا، الحقل الذي يحدد عدد الدواسات الموجودة في نوع معين من الدراجات يمكن أن يتم إعلانه باستعمالstatic ، بما أن نفس عدد الدواسات ينطبق على جميع الكائنات. يمكن إنشاء مثل هذا الحقل باستعمال الشيفرة التالية

```
static int numGears = 6;
```

بالإضافة إلى ذلك، يمكن استعمال الكلمة المفتاحية final للإشارة إلى أن عدد الدواسات لن يتغير أبدا.

- المتغيرات المحلية مثل ما تقوم الكائنات بتخزين حالتها في الحقول، فإن الطرق تُخَزِّن أحيانا حالتها المؤقتة في *متغيرات محلية* .طريقة إعلان المتغيرات المحلية مشابهة لطريقة إعلان الحقول، مثلا

```
int count = 0;
```

لا توجد كلمة مفتاحية تُحَدِّد أن متغيرا ما هو متغير محلي؛ تحديد ذلك يعتمد أساسا على المكان الذي تم فيه إعلان المتغير - والذي يوجد بين المعقوفات التي تحدد بداية ونهاية الطريقة. وبالتالي، فإن المتغيرات المحلية تكون مرئية فقط داخل الطرق التي أُعْلنت داخلها؛ ولا يمكن الوصول إليها من أي مكان آخر في الفئة.

- المُعْطيات سبق ورأيت أمثلة لمعطيات، في الفئة Bicycle والطريقة main في البرنامج ."Hello World!" تذكر أن توقيع الطريقة main هو

```
public static void main(String[] args)
```

هنا، المتغير args يُمَثل مُعْطى هذه الطريقة. الأمر المهم الذي يجب تذكره، هو أن المعطيات تُصَنَّف ك"متغيرات" وليس ك"حقول". هذا ينطبق أيضا على كل البنيات التي تقبل المعطيات (مثل الconstructors ومعالجات الإستثناءات) التي سنراها فيما يلي من الدروس.

فيما يلي، سنتبع القواعد التالية عندما نتكلم عن المتغيرات والحقول: إذا كنا نتحدث عن "الحقول بصفة عامة"، (مع استثناء المتغيرات المحلية والمُعْطيات)، سنقول فقط "حقول". إذا كان الكلام ينطبق على "كل ما سبق"، سنستخدم فقط كلمة "متغيرات". إذا كان السياق يجبرنا على التمييز، سنستخدم عبارات محددة (متغير مخلي، حقل ...static). يمكن أيضا في بعض الأحيان استعمال المصطلح "عضو member. "الحقول، الطرق والفئات الداخلية الخاصة بفئة ما، كلها تعتبر *أعضاء*.

التسمية

كل لغة برمجية لها قواعدها الخاصة فيما يخص أنواع الأسماء المسموح بها، والجافا ليست استثناء. يمكن تلخيص قواعد تسمية المتغيرات في ما يلي:

- أسماء المتغيرات حساسة لحالة الحرف case-sensitive. إسم المتغير يمكن أن يكون أي معرف مسموح به -- سلسلة غير محدودة الطول من حروف ال unicode والأعداد، تبتدأ بحرف، رمز الدولار $ أو حرف التسطير السفلي "_". لكن الإجماع هو على عدم استعمال ال $ و "_" في بداية الإسم، والإكتفاء بالحروف .بالإضافة إلى ذلك، فإن هناك إجماعا على عدم استعمال الرمز $ بالمرة. يمكن أن تجد أحيانا بعض الأسماء المُوَلّدة تلقائيا تختوي على الرمز $، لكن لا يجدر بك استعماله عند تسمية متغيراتك. هناك أيضا إجماع آخر يخص رمز التسطير السفلي، فبالرغم من كون استعماله في بداية الأسماء مسموحا به تقنيا، إلا أنه لا ينصح بذلك. بالنسبة للفراغات فغير مسموح بها.
- الرموز التي تلي لحرف الأول يمكن أن تكون حروفا، أرقاما، الرمز $ أو رمز التسطير السفلي "_". الإجماع (والمنطق) ينطبقان على هذه القاعدة أيضا .عند اختيار إسم لمتغيراتك، إستعمل كلمات كاملة بدل اختصارات غير مفهومة .هذا يجعل الشيفرة سهلة القراءة والفهم، وفي كثير من لأحيان، ذلك يجعل شيفرتك ذاتية التوثيق؛ فمثلا، الحقول المسماة cadence, speed و gear تُعْتبر أكثر تعبيرا من الأسماء المختصرة، مثل c, s و g. يجب أيضا أن تأخذ في الحسبان أن لا يكون الإسم أحد [الكلمات المفتاحية أو الكلمات المحجوزة]

.

- إذا كان الإسم الذي اخترت يحتوي على كلمة واحدة فقط، إجعل كل الحروف صغيرة.lowercase letters إذا كان يختوي على أكثر من كلمة، إجعل الخرف الأول من كل كلمة بعد الأولى كبيراCapital ، الأسماء gearRatio و currentGearتعتبر أمثلة لهذه القاعدة.

إذا كان المتغير يختوي على قيمة ثابثة، مثلا

```
static final int NUM_GEARS = 6
```

فإن القاعدة تتغير. حيث يجب جعل كل الحروف كبيرة، والفصل بين كل كلمة باستعمال التسطير السفلي. هناك إجماع على عدم استعمال رمز التسطير السفلي في أي مكان آخر .

الأصناف البدائية للبيانات

لغة البرمجة جافا تُعتبر لغة ساكنة التصنيفstatically-types ، ما يعني أنه يجب علينا تعريف كل المتغيرات قبل أن نتمكن من استعمالها. ما يعني تحديد نوع المتغير وإسمه، كما رأينا في الأمثلة السابقة:

```
int gear = 1;
```

بكتابتنا لهذا السطر فإننا نقول للبرنامج أن هناك حقلا إسمه"gear" ، يخزن قيمة عددية، وقيمته الأولية هي "1". يتم تحديد نوع بيانات متغير ما انطلاقا من القيم التي يمكن أن يُخَزّنها، ومن العمليات التي يمكن تنفيذها عليه.

بالإضافة إلىint ، لغة البرمجة جافا تدعم سبعة *أصناف بيانات بدائية* أخرى. الأصناف البدائية محددة مسبقا من طرف اللغة ومُسَمَّاة بكلمة محجوزة. القيم البدائية لا تتشارك حالتها مع القيم البدائية الأخرى .الأصناف البدائية للبيانات المدعومة من طرف لغة البرمجة جافا هي:

- byte: صنف البيانات byte يمثل عددا صحيحا حجمه 8 بايت، signed، ويستخدم نظام ال two's complement. قيمته الدُّنيا هي 128- وقيمته القصوى تصل إلى 127. الصنف byte يمكن أن يكون مفيدا للحفاظ على الذاكرة عند استعمال المصفوفات الكبيرة، حيث تكون هناك حاجة لاقتصاد الذاكرة. يمكن أيضا استعمال هذا الصنف مكان int حيث يمكن لحدود القيم الممكنة أن تساعد في توضيح الشيفرة؛ كون مجال المتغير محدودا يمكن أن يكون شكلا من أشكال التوثيق.
- short: صنف البيانات short يمثل عددا صحيحا حجمه 16 بايت، signed، ويستخدم نظام ال two's complement.قيمته الدُّنيا هي 32,768- وقيمته القصوى تصل إلى 32,767. كما هو الحال بالنسبة لbyte ، يمكن استعمال short لاقتصاد الذاكرة في المصفوفات الكبيرة، في المواقف التي يكون فيها الحفاظ على الذاكرة ضروريا.
- int: صنف البيانات int يمثل عددا صحيحا حجمه 32 بايت، signed ، ويستخدم نظام ال two's complement. قيمته الدُّنيا هي 2,147,483,648- وقيمته القصوى تصل إلى 2,147,483,647. بالنسبة للقيم الصحيحة غير الكسرية، هذا الصنف يكون عادة الإختيار الإفتراضي، إلا إذا كان هناك سبب ما (كما رأينا سابقا) لعدم استعماله. هذا الصنف سيكون غالبا كبيرا كفاية لاحتواء الأعداد التي سيستعملها برنامجك، لكن إذا احتجت مجالا أكبر من القيم، استعملlong.
- long: صنف البيانات long يمثل عددا صحيحا حجمه 64 بايت، signed، ويستخدم نظام ال two's complement. قيمته الدُّنيا هي 9,223,372,036,854,775,808- وقيمته القصوى تصل إلى 9,223,372,036,854,775,807. إستعمل هذا الصنف عندما تحتاج إلى قيم أكبر من تلك التي يوفرها الصنفint.
- float: صنف البيانات float يمثل عددا حجمه 32 بايت، ذو فاصلة عائمة، أحادي الدقة حسب المعيار IEEE 754. مجال هذا الصنف يفوق نطاق هذا الدرس، لكنه مُحدد في الجزء 4.2.3 من مواصفات لغة الجافا. كما هو الحال بالنسبة ل byteو short ، استعمل) float بدل double) عندما تحتاج الحفاظ على الذاكرة عند التعامل مع مصفوفات كبيرة من الأعداد عائمة الفاصلة. هذا الصنف لا يجب أبدا أن يُستعمل للقيم الدقيقة مثل المبالغ المالية، بل يجب استعمال الفئة BigDecimal. في الأعداد وسلسلات الحروف سنتكلم عن BigDecimal وفئات مفيدة أخرى متوفرة في منصة الجافا.
- double: صنف البيانات double يمثل عددا حجمه 64 بايت، ذو فاصلة عائمة، أحادي الدقة حسب المعيار IEEE 754.مجال هذا الصنف يفوق نطاق هذا الدرس، لكنه مُحدد في الجزء 4.2.3 من مواصفات لغة الجافا. بالنسبة للقيم العشرية، هذا الصنف هو الإختيار الإفتراضي. لا يجب أبدا استعمال هذا الصنف للتعامل مع القيم الدقيقة مثل المبالغ المالية.
- boolean: صنف البيانات boolean يقبل قيمتين فقط true :و false. إستعمل هذا الصنف لتتبع الشروط صحيح/خطأ. هذا الصنف يمثل معلومة مخزنة في bit واحد، لكن "حجمه" ليس بشيء يمكن تحديده بدقة.
- char: صنف البيانات char هو حرف unicode بحجم 16 بايت. قيمته الدنيا هي) '\u0000' أو 0) وقيمته القصوى هي '\uffff' (أو 65,535).

بالإضافة إلى الأصناف البدائية الثماني المذكورة في الأعلى، لغة البرمجة جافا تقدم دعما لسلسلات الحروف عبر الفئة java.lang.String. وضع سلسلة حروف بين علامتي إقتباس مزدوجتين ينشئ كائن String جديد؛ مثلا

```
String s = "هذه سلسلة حروف";
```

الكائنات String *غير قابلة للتعديل*، ما يعني أن قيمتها لا يمكن أن تتغير بعد أن يتم إنشائها. الفئة String ليست تقنيا صنف بيانات بدائيا، لكن بالنظر للدعم الخاص الذي توفره لها اللغة، يمكن أن تتكون لديك هذه الفكرة. سنتكلم أكثر عن الفئة String في الكائنات البسيطة.

القيم الإفتراضية

ليس من الضروري أن نقوم دائما بتحديد قيمة حقل ما عند إعلانه. الحقول التي يتم إعلانها بدون تهيئتها سيتم إعطائها قيمة افتراضية من طرف المُجمع compiler. بصفة عامة، هذه القيمة ستكون صفر أوnull ، حسب الصنف. لكن الإعتماد على هذه القيم الإفتراضية يعتبر أسلوبا سيئا للبرمجة.

الجدول التالي يعطي القيم الإفتراضية لبعض الأصناف.

صنف البيانات	القيمة الإفتراضية (بالنسبة للحقول)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
String (أو أي كائن)	null
boolean	false

بالنسبة للمتغيرات المحلية فالأمر مختلف، فالمجمع لا يعطي أبدا قيمة افتراضية لمتغير محلي غير مُهَيَّء. إذا لم يكن بإمكانك تهيئة متغير محلي عند إعلانه، تأكد من تحديد قيمته قبل أي محاولة لاستعماله. إستعمال متغير محلي غير مهيئ ينتج عنه خطأ عند التجميع.

Literals

ربما لاحظت أننا لا نستعمل الكلمة المفتاحية new عندما نهيء متغيرا ذا صنف بدائي. الأصناف البدائية هي أصناف بيانات مبنية في داخل اللغة، وليست كائنات نُنشئها انطلاقا من فئة literal .هو كل قيمة ثابتة تظهر في الشيفرة المصدرية؛ ال literals تُضاف مباشرة إلى الشيفرة دون القيام بأي حسابات .كما يظهر في الأسفل، يمكن إعطاء literal إلى متغير من صنف اساسي:

```
boolean result = true;

char capitalC = 'C';

byte b = 100;

short s = 10000;

int i = 100000;
```

الأصناف الصحيحة غير الكسرية (byte, short, int, & long) يمكن التعبير عنها باستعمال نظام الأعداد العشري، الثماني أو الست عشري. النظام العشري هو النظام الذي تستعمله كل يوم؛ وهو يستعمل 10 أرقام، من 0 إلى 9 .النظام الثماني أساسه 8، أي أننا نستطيع استعمال الأرقام من 0 إلى 7 فقط .النظام الست عشري أساسه 16، و الأرقام المتوفرة فيه هي الأرقام من 0 إلى 9 والأحرف من A إلى F. بالنسبة للبرمجة العامة التوجه، النظام العشري سيكون غالبا النظام الوحيد الذي ستستعمل، لكن إذا كنت بحاجة لاستعمال النظام الثماني أو الست عشري، الأمثلة التالية تبين الطريقة الصحيحة لاستعمالهما .

البادئة 0 تحدد النظام الثماني، بينما 0 xتحدد النظام الست عشري.

```
int decVal = 26;        // العدد 26 في النظام العشري
int octVal = 032;       // العدد 26 في النظام الثماني
```

```
int hexVal = 0x1a;     // العدد 26 في النظام الست عشري
```

الأصناف ذات الفاصلة العائمة (float و double) يمكن أن تُكتب باستعمال E أو e (الكتابة العلمية)، Fأو f أي (32 - bit float literal) و D أو d التي تعني (64- double literal bit ؛ هذا النصرف افتراضي، لذلك يتم تجاهل هده الكتابة) .

```
double d1 = 123.4;

double d2 = 1.234e2; // نفس قيمة d1 لكن بالكتابة العلمية ،

float f1  = 123.4f;
```

ال literals من صنف char و String يمكن أن تحتوي على أي حرف .Unicode (UTF-16) إذا كان المحرر ونظام الملفات يسمحان بذلك، يمكنك كتابة هذه الحروف مباشرة داخل الشيفرة. إذا لم تكن كذلك، يمكنك استعمال "Unicode escape" مثل '\u0108' (حرف C مع circumflex) ، أو "S\u00ED se\u00F1or" وتعني (Sí Señor بالإسبانية). استعمل دائما علامة الإقتباس الأحادية (') بالنسبة لchar literals، وعلامة الإقتباس المزدوجة (") بالنسبة ل String literals. ال Unicode escapeيمكن أن تُستعمل في أماكن أخرى من البرنامج (أسماء الحقول مثلا)، ليس فقط في ال char و String literals.

لغة الرمجة جافا تدعم أيضا بعض ال escape sequences الخاصة بالنسبة لل literals من صنف char و String:

\b (backspace),
\t (tab),
\n (line feed),
\f (form feed),
\r (carriage return),
\" (double quote),
\' (single quote),
\\ (backslash).

يوجد أيضا literal خاص : null والذي يمكن أن يُستعمل كقيمة لأي صنف reference. يمكن لأي متغير أن يأخد القيمةnull ، باستثناء المتغيرات من الأصناف الأساسية. ليس هناك الكثير مما يمكن فعله بقيمة null عدا التحقق من وجودها، لذلك فهي تستعمل كثيرا في البرامج كمُؤشر يبين أن كائنا ما غير متوفر.

أخيرا، هناك أيضا نوع خاص من ال literals يسمى*class literal* ، نحصل عليه بضم إسم صنف مع".class" ؛ مثلا : String.class.هذا يشير إلى الكائن (من صنف class) الذي يمثل الصنف بذاته.

--
-

البرمجة الكائنية التوجه - OOP

ما هو الكائن؟

الكائنات (Objects) هي المفتاح لفهم البرمجة *الكائنية التوجه* .ألق نظرة حولك وستجد الكثير من الأمثلة الملموسة لكائنات (Objects)كلبك، مكتبك، تلفازك، دراجتك... :

الكائنات الموجودة في أرض الواقع تتشارك في خاصيتين : كلها لديها *حالة و سلوك* .الكلاب لها حالة (إسم، لون، ذرية، جائع) وسلوك (نباح، جلب، تحريك الذيل .(الدراجة أيضا لديها حالة (الدولاب الحالي، السرعة الحالية) وسلوك (تغيير الدولاب، الفرملة). التعرف على حالة وسلوكات الكائنات الموجودة في أرض الواقع هي طريقة جيدة لفهم مبادئ البرمجة الكائنية التوجه.

خذ الآن دقيقة لملاحظة الكائنات التي حولك. لكل كائن تراه، اسأل نفسك سؤالين: "ما هي الحالات الممكنة التي يمكن أن يكون عليها؟" و "ما هي السلوكات التي ممكن أن يقوم بها؟". تأكد من كتابة ملاحظاتك. وأنت تقوم بذلك، ستلاحظ أن الكائنات المحيطة بنا تتفاوت من حيث التعقيد؛ فمصباحك المكتبي يمكن أن يتوفر على حالتين فقط (مفتوح، مقفل) وسلوكين (فتح، إقفال(، بينما جهاز الراديو يمكن أن يتوفر على حالات إضافية (مفتوح، مقفل، درجة الصوت، المحطة الحالية) وسلوكات (فتح، إقفال، زيادة الصوت، تخفيض الصوت، البحث عن محطة، تحديد المحطة). يمكن أن تلاحظ أيضا أن بعض الكائنات يمكن أن تحتوي على كائنات أخرى. كل هذه الملاحظات تنطبق على البرمجة الكائنية التوجه.

تعتبر الكائنات البرمجية مشابهة للكائنات الحقيقية من حيث المبدأ: كلاهما يتجلى في حالة وسلوكات. الكائن يخزن حالته في *حقول* (fields) (تسمى متغيرات في لغات برمجية أخرى) ويعرض تصرفاته من خلال *طرق* (methods) (تسمى دوال (functions) في لغات برمجية أخرى). الطرق تعمل على الحالة الخاصة بالكائنات، وتشكل الآلية الأساسية للتواصل بين الكائنات .إخفاء الحالة واشتراط أن يكون كل تفاعل مع الكائن يمر عبر الطرق الخاصة به، يعرف ب *تغليف البيانات*(Data encapsulation) ، وهو مبدأ أساسي في البرمجة الكائنية التوجه.

لنأخذ الدراجة كمثال:

عندما نحدد الحالة (السرعة الحالية، الإيقاع الحالي للدواسة، الدولاب الحالي) ونوفر الطرف التي تستطيع تغيير هذه الحالة، فإن الكائن يبقى متحكما في الطريقة المتاحة لاستعماله من أي عنصر خارجي. مثلا، إذا كانت الدراجة تتوفر فقط على 6 دواليب، فإن الطريقة التي تقوم بتغيير الدولاب يمكنها أن ترفض أي قيمة أصغر من 1 وأكبر من 6.

تجميع الكود في كائنات برمجية يوفر عدة مزايا، من بينها :

1. ال Modularity : الكود المصدري لكائن ما يمكن أن تتم كتابته وصيانته بطريقة مستقلة عن الكائنات الأخرى. بعد إنشائه، يمكن تمرير الكائن بسهولة في أنحاء النظام.
2. إخفاء المعلومات: بفضل التعامل مع الطرق الخاصة بالكائن فقط، يبقى التطبيق (implementation) مخفيا عن العالم الخارجي.
3. إعادة استعمال الكود: إذا كان كائن ما موجود مسبقا (كتبه مبرمج آخر مثلا)، يمكنك استعمال هذا الكائن داخل برنامجك. هذا يسمح للمختصين بعمل implement/test/debug لكائنات معقدة وموجهة لغرض محدد، مما يجعلك تشتغل بها بكل ثقة داخل الكود.
4. سهولة ال :Pluggability and debugging إذا اتضح أن كائنا يخلق مشاكل، يمكن إزالته وربط كائن آخر ليشغل مكانه. هذا الأمر مشابه لإصلاح المشاكل الميكانيكية في الواقع. إذا تعطل الصاعق، فإنك تغيره لوحده، ولا تغير كل المحرك.

ما هي الفئة؟

في أرض الواقع، ستجد الكثير من الكائنات من نفس النوع. يوجد هناك ربما المئات من الدراجات، كلها من نفس المصنع ونفس النموذج. كل دراجة تم بناؤها انطلاقا من نفس المخطط وبالتالي تحتوي على نفس المكونات. في إطار البرمجة الكائنية التوجه، نقول أن الدراجة عبارة عن ***instance*** *لفئة الكائنات* المسماة بالدراجات. *الفئة* **(Class)** هي المخطط الذي نبني انطلاقا منه الكائنات.

الفئة التالية **Bicycle** هي تطبيق ممكن لدراجة.

```
class Bicycle {

	int cadence = 0;
	int speed = 0;
	int gear = 1;

	void changeCadence(int newValue) {
		cadence = newValue;
	}

	void changeGear(int newValue) {
		gear = newValue;
	}

	void speedUp(int increment) {
		speed = speed + increment;
	}

	void applyBrakes(int decrement) {
		speed = speed - decrement;
	}

	void printStates() {
		System.out.println("cadence:"+cadence+" speed:"+speed+"
gear:"+gear);
	}
}
```

طريقة الكتابة بالجافا ستبدو جديدة عليك، لكن تصميم هذه الفئة يعتمد على ما تكلمنا عنه سابقا بخصوص الدراجة ككائن. الحقول **speed** ، **cadence** و **gear** تمثل حالة الكائن ، والطرق**speedUp** ، **changeGear** ، **changeCadence** ... تحدد طريقة تفاعله مع العالم الخارجي.

ربما لاحظت أن الفئة Bicycle لا تحتوي على الطريقةmain ، وذلك لأنها ليست برنامجا كاملا؛ بل فقط مخطط للدراجات التي يمكن أن*تستعمل* في برنامج ما. مسؤولية إنشاء واستعمال كائنات Bicycle جديدة تقع على عاتق فئات أخرى في برنامجك.

هذا مثال ينشئ كائنين Bicycle وينادي على الطرق الخاصة بها:

```
class BicycleDemo {
        public static void main(String[] args) {

                // إنشاء كائنين Bicycle مختلفين
                Bicycle bike1 = new Bicycle();
                Bicycle bike2 = new Bicycle();

                // المناداة على الطرق الخاصة بالكائنات المنشأة
                bike1.changeCadence(50);
                bike1.speedUp(10);
                bike1.changeGear(2);
                bike1.printStates();

                bike2.changeCadence(50);
                bike2.speedUp(10);
                bike2.changeGear(2);
                bike2.changeCadence(40);
                bike2.speedUp(10);
                bike2.changeGear(3);
                bike2.printStates();
        }
}
```

مخرجات هذا المثال تعطي القيم النهائية الخاصة ب: إيقاع الدواسة، السرعة، والدولاب بالنسبة للدراجتين:

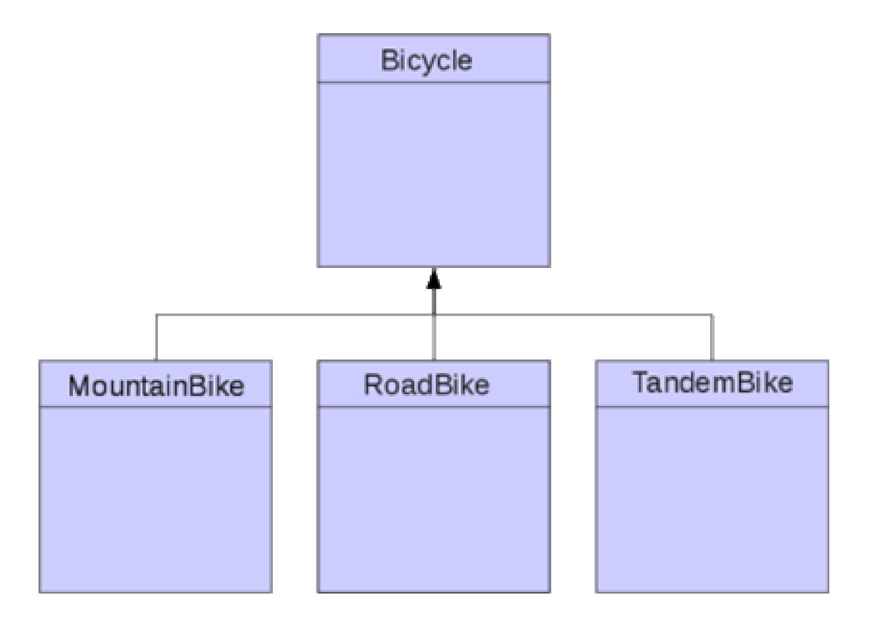
cadence:50 speed:10 gear:2

cadence:40 speed:20 gear:3

ما هي الوراثة؟

نجد أحياناً أنواعاً مختلفة من الكائنات لديها بعض القواسم المشتركة .مثلا، الدراجات الجبلية، دراجات الطريق، والدراجات الترادفية، كلها تتشارك خصائص الدراجات (السرعة الحالية، الإيقاع الحالي للدواسة، الدولاب الحالي)، بالإضافة إلى ذلك، فكل نوع لديه خصائص إضافية تجعله مختلفا عن الأنواع الأخرى: الدراجات الترادفية لديها مقعدين ومقودين؛ دراجات الطريق لديها مقود متجه للأسفل؛ بعض الدراجات الجبلية تمتلك سلسلة بحلقة إضافية، ما يعطيها **a lower gear ratio**.

البرمجة الكائنية التوجه تتيح للفئات *وراثة* الحالة والسلوكات المشتركة من فئات أخرى. في المثال التالي، الفئة **Bicycle** ستصبح *الفئة الأم* لـ**MountainBike** ، **RoadBike**، و **TandemBike.** في لغة البرمجة جافا، يسمح لكل فئة بامتلاك فئة أم واحدة فقط، وكل فئة أم يسمح لها بعدد لا منتهي من *الفئات الفرعية*:



طريقة إنشاء فئة فرعية بسيطة. عند تعريف الفئة، استعمل الكلمة المفتاحية **extends** ، متبوعة بإسم الفئة الأم:

```
class MountainBike extends Bicycle {

    // نعرف هنا الحقول والطرق الجديدة الخاصة بالدراجات الجبلية

}
```

هذا يعطي للفئة **MountainBike** نفس الطرق والخصائص الموجودة في**Bicycle** ، مما يسمح بالتركيز على الميزات التي تنفرد بها الدراجات الجبلية، هذا يجعل شيفرة الفئات الفرعية سهلة القراءة.

ما هي الواجهة؟

كما رأينا سابقا، فالكائنات تتفاعل مع العالم الخارجي بواسطة الطرق (methods) التي تعرضها. إذن فالطرق تمثل *واجهة* الكائن مع العالم الخارجي(interface)؛ مثلا، الأزرار الموجودة في مقدمة التلفاز هي الواجهة بينك وبين الأسلاك الكهربائية الموجودة في الجهة الأخرى من الغطاء البلاستيكي. عندما تريد تشغيل أو إغلاق الجهاز، فإنك تضغط على زر التشغيل.

في أغلب الأحيان، تكون الواجهة عبارة عن مجموعة من الطرق المرتبطة، ذات جسم خالي. السلوك الخاص بالدراجة، إذا ما تم تعريفه على شكل واجهة، يمكن أن يكون بالشكل التالي:

```
interface Bicycle {

       void changeCadence(int newValue);

       void changeGear(int newValue);

       void speedUp(int increment);

       void applyBrakes(int decrement);
}
```

لتطبيق هذه الواجهة، يجب أن يكون إسم الفئة مغايرا (إضافة إسم الماركة مثلا، لنقل ACMEBicycle) ، ويجب استعمال الكلمة المفتاحية implements عند تعريف الفئة:

```
class ACMEBicycle implements Bicycle {

   // نفس الشيفرة التي استعملنا في مثال الدراجة في الدروس السابقة

}
```

تطبيق واجهة يسمح للفئة بأن تصبح أكثر التزاما بالسلوك الذي تعد بتوفيره. الواجهات تمثل عقدا بين الفئة والعالم الخارجي، وهذا العقد يتم تأكيده عند عملية البناء من طرف المترجم .(compiler) إذا كانت فئتك تسعى لتطبيق واجهة ما، فيجب أن تحتوي على كل الطرق المعرفة داخل الواجهة، حتى تتم عملية الترجمة (compilation) بدون أخطاء .

ملحوظة: لكي تتم ترجمة الفئة ACMEBicycle بنجاح، يجب إضافة الكلمة المفتاحية public في بداية كل الطرق الخاصة بالواجهة والتي يتم تطبيقها. ستعلم أسباب ذلك لاحقا عندما نتحدث عن الفئات والكائنات و الواجهات والوراثة.

ما هي الحزمة؟

الحزمة (package) عبارة عن مجال إسم(namespace) ، وهدفها تنظيم مجموعة من الفئات والواجهات ذات العلاقة. يمكن مقارنة فكرة الحزمة بمجلدات مختلفة موجودة في حاسوبك، مجلد خاص بالملفاتHTML ، مجلد آخر خاص بالصور، وآخر يحتوي على البرامج.

لأن البرامج المكتوبة بلغة الجافا يمكن أن تحتوي على المئات وربما*الآلاف* من الفئات، فإن المنطق يحتم علينا إبقاء كل شيء منظما، بوضع الفئات والواجهات ذات العلاقة داخل حزم.

منصة الجافا تقدم مكتبة فئات عملاقة (مجموعة من الحزم) مناسبة للإستعمال داخل برنامجك. هذه المكتبة تعرف ب "واجهة برمجة التطبيقات (Application Programming Interface) "أو "API" باختصار. هذه الحزم تمثل المهام الأكثر ارتباطا بالبرمجة الشاملة .(general-purpose programming) على سبيل المثال، كائن String يحتوي على حالة وسلوك سلسلات الحروف؛ كائن File يتيح للمبرمج إنشاء، مسح، تفتيش، مقارنة، أو تعديل ملف ما بكل سهولة؛ كائن Socket يسمح بإنشاء واستعمال الsockets network ؛ كائنات عديدة خاصة بواجهة المستخدم الرسومية تتحكم بالأزرار وخانات التأشير (checkbox)وبكل ما يتعلق بواجهات المستخدم الرسومية.
هناك الآلاف من الفئات التي يمكننا أن نختار من بينها، هذا يسمح لك، المبرمج، بالتركيز على طريقة تصميم برنامجك، بدل التركيز على البنية التحتية اللازمة لتشغيله.

ال Java Platform API Specification تحتوي على لائحة بجميع الحزم، الواجهات، الفئات، الحقول والطرق التي توفرها منصة الجافة 6، النسخة المعيارية.(StandardEdition)
إفتح الصفحة في المتصفح، وأضفها إلى المفضلة. كمبرمج، سيكون ذلك الموقع بالنسبة لك بمثابة أهم مرجع لتوثيق لغة الجافا.

أسئلة وتمارين

أسئلة

1. الكائنات في العالم الحقيقي تتوفر على ـ و ـ.
2. يتم تخزين حالة كائن برمجي في ـ.
3. يتم عرض سلوك كائن برمجي عبر ـ.
4. إخفاء البيانات الداخلية عن العالم الخارجي، والوصول إليها فقط باستعمال الطرق العلنية يعرف ب ـ.
5. مخطط كائن برمجي يسمى ـ.
6. السلوكات المشتركة يمكن تعريفها في ـ ويمكن وراثتها من طرف ـ باستعمال الكلمة المفتاحية ـ.
7. مجموعة من الطرق بدون جسم تسمى ـ.
8. مجال الإسم (namespace) التي تنظم الفئات والواجهات حسب وظيفتها يسمى ـ.
9. كلمة API تعني ـ.

تمارين

1. أكتب فئة جديدة لكل واحد من كائنات العالم الحقيقي التي لاحظت في بداية هذا الدرس. إرجع إلى الفئة Bicycle إذا نسيت طريقة الكتابة.
2. لكل واحدة من الفئات التي كتبت أعلاه، أنشئ واجهة تحدد سلوكها، ثم أجبر فئتك على تطبيقها. أهمل طريقة أو طريقتين وحاول القيام بعملية التجميع. ما هو الخطأ الناتج؟

الأجوبة

أجوبة الأسئلة

1. الكائنات في العالم الحقيقي تتوفر على ***حالة*** و ***سلوك***.
2. يتم تخزين حالة كائن برمجي في ***حقول***.
3. يتم عرض سلوك كائن برمجي عبر ***طرق***.
4. إخفاء البيانات الداخلية عن العالم الخارجي، والوصول إليها فقط باستعمال الطرق العلنية يعرف ب ***تغليف البيانات***.
5. مخطط كائن برمجي يسمى ***فئة***.
6. السلوكات المشتركة يمكن تعريفها في ***الفئة الأم*** ويمكن وراثتها من طرف ***فئة فرعية*** باستعمال الكلمة المفتاحية ***extends***.
7. مجموعة من الطرق بدون جسم تسمى ـ.
8. مجال الإسم (namespace) الذي ينظم الفئات والواجهات حسب وظيفتها يسمى ***حزمة***.
9. كلمة API تعني ***Application Programming Interface***: واجهة برمجة التطبيقات.

حل التمارين

1. الأجوبة ستكون مختلفة، حسب الكائنات التي استعرضتم.
2. الأجوبة ستكون مختلفة هنا أيضا، لكن بالنسبة لرسالة الخطأ، فستشير إلى الطرق المنتظرة، والتي تم إهمالها.